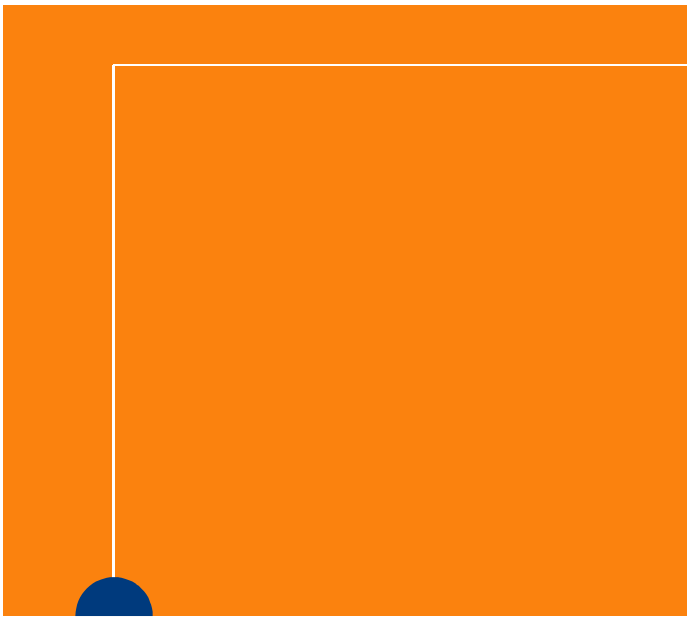
MOTOROLA
intelligence everywhere™

Using Serial on Motorola J2ME handsets

April 25, 2005



WHITE PAPER

Using Serial on Motorola J2ME handsets

By

Motocoder staff

Serial connection is a commonly used connection to exchange data, and often used in real time control and management, also serial I/O is the common interface in a PC application system. For many Motorola J2ME handsets, to use serial I/O is possible, you can communicate with other devices and applications within J2ME. This guide outlines the use of serial I/O in J2ME and offers some notes that to which you need to pay attention.

Serial introduction

In general, J2ME serial connectivity can be done via the GCF (Generic Connection Framework) by creating a CommConnection (or StreamConnection). GCF was originally defined to rely on the J2ME platform's Connected Limited Device Configuration (CLDC), version 1.0, because the familiar J2SE java.net and java.io APIs were considered too large to fit into the constrained memory available in mobile devices. And GCF is a straightforward hierarchy of interfaces and classes to create connections (such as HTTP, datagram, or streams) and perform I/O. MIDP2.0 has defined an optional interface CommConnection (javax.microedition.io.CommConnection). Because this is not a mandatory connection type, the handset must provide an implementation of CommConnection in order for this to work. Here is a figure for GCF that lists stream connection and com connection interface.

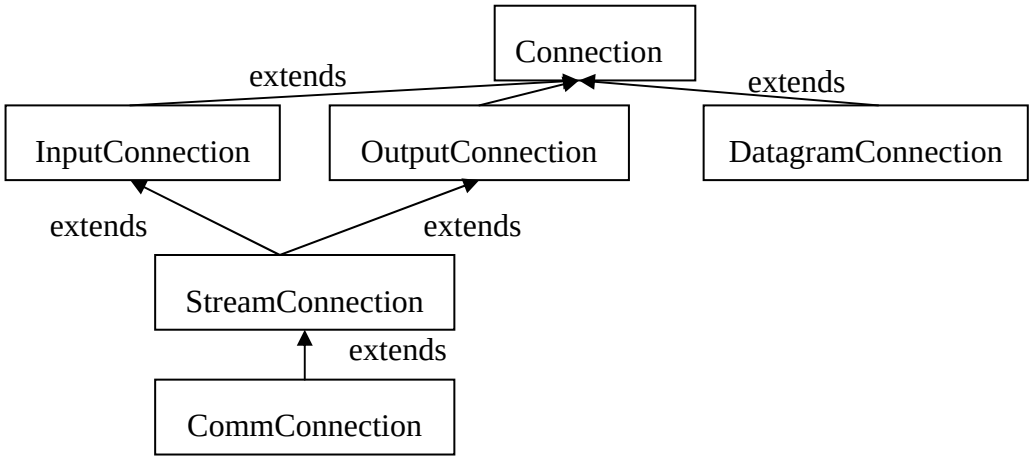


Figure1: Serial related Generic Connection Framework Interface Hierarchy

StreamConnection is the super interface of CommConnection. For MIDP1.0 device you can use StreamConnection without the support of CommConnection.

Table1 summarizes serial I/O related GCF connection types:

Table1: GCF Serial Connection Types, by Profile

GCF Interfaces	Vendors Must Support?	CLDC 1.0, 1.1	MIDP 1.0	MIDP 2.0
StreamConnection	Yes	X	X	X
CommConnection	No			X

X: include the interface

Developing Applications using serial I/O

Using serial I/O you can develop applications to communicate with other devices such as a GPS receiver. You can connect to modems or bar code scanners etc. You can also capture video snapshot or audio in J2ME from the handset and use serial I/O to transfer them to your PC. In GCF the Connector class is the connection factory. It creates and opens an instance of the appropriate subtype of Connection based on the URL you specify, if that type is supported.

The Connector class defines the following open methods:

- open(String url)
- open(String url, int mode)
- open(String url, int mode, boolean timeouts)

The parameters used in open method:

- url is the URL that describes the connection type to create.
- mode is the connection mode to use - READ, WRITE, or READ_WRITE (the default).
- timeouts is a flag to indicate that the caller wants to be notified about timeout exceptions (InterruptedIOException), if the underlying connection implements notification. (the default is false.)

In Connector.open(String url, int mode, boolean timeouts), The url's scheme (defined in RFC 2396) must be for serial connection:

```
comm:<port identifier>[<optional parameters>]
```

A comm port is accessed using a GCF string with an explicit port identifier and embedded configuration parameters, each separated with a semi-colon (;).

For example:

```
 (CommConnection)Connector.open("comm:COM0;baudrate=19200", Connector.READ_WRITE,
false);
```

The optional parameters, includes parameters such as the baudrate (the speed of the port). The baudrate can be set which is platform dependent, if unsupported and baudrate is set, an IOException: Unsupported baudrate is thrown. It can be left unset and the system will assign a baudrate for it automatically.

If the final parameter timeouts, is passed to Connector.open() is false, you will not get notified when the phone can't make a serial connection. The default value is false. If you want to get notification when the connection times out, you should use "true" as the timeouts parameter.

The following example shows how a CommConnection would be used to discover available comm ports.

```
String port;
String ports = System.getProperty("microedition.commports");
```

```
int comma = ports.indexOf(',');
if (comma > 0) {
    // Parse the first port from the available ports list.
    port = ports.substring(0, comma);
} else {
    // Only one serial port available.
    port =ports;
}
```

Below is a code example showing how you can use serial I/O API to open a connection and read/write to it.

```
//using commconnection since midp2.0
CommConnection commConn =
(CommConnection)Connector.open("comm:COM0;baudrate=115200", Connector.READ_WRITE,
true);
InputStream iStream = commConn.openInputStream();
OutputStream oStream = commConn.openOutputStream();
String sMessage = "send";
oStream.write(sMessage.getBytes());
Thread tRead = new Thread() {
    public void run() {
        while (true) {
            StringBuffer sb = new StringBuffer();
            while(true) {
                int c = iStream.read();
                if(c != '\r') {
                    sb.append((char)c);
                }
                else
                    break;
            }
            try {
                sleep(1000);
            }
            catch (InterruptedException e) {}
        }
    }
};
tRead.start();
```

For some handsets that do not support CommConnection, you can use StreamConnection in place of CommConnection with the same result:

```
StreamConnection commConn =
(StreamConnection)Connector.open("comm:COM0";baudrate=115200");
```

For reading and writing to serial port, you can do them in a separate thread so that it will not block the main thread.

Motorola SDK Emulator notes

Sometimes you may want to test serial I/O MIDlet in an emulator first, using serial in an SDK emulator maybe different from the actual handset. Here lists some notes for using it.

Different emulators use a differently implemented class for comm connection. Let's take Motorola SDK4.4 as an example, for EmulatorA.3: c975, c980, v980, e1000 it fails to open a serial connection, the lib class does not support serial I/O connection. And other emulators are ok. They use a different lib class which supports comm connection. For EmulatorA.1, the connection open format is different. Add "/" to the comm url when opening a serial connection, for example:

```
(CommConnection)Connector.open("comm:/COM1;baudrate=19200");
```

For EmulatorM.1 and EmulatorM.3, you can use them for serial I/O test. They use a differently implemented class, the comm url for opening a serial connection does not need to be changed. You can see the release notes of different emulators for details.

Notes of using serial IO

Although serial I/O seems easy to use, there are still some notes to which you need to pay attention.

1. Here is the process or guide for using serial connection code, after opening a serial connection (the Connector.open() is executed) then plug in the serial cable. This means that you have to start the application without the cable connected, and as soon as you have executed the code that tries to establish a serial connection, you insert the cable. (This process is commonly used for most of our J2ME handsets, except some Ezx handsets such as E680. For E680 you can have the USB cable connected all through the running of your MIDlet).

For using serial in T720i, please refer to the FAQ listed below in note 8. T720i uses the Motorola Proprietary Serial API and is slightly different to the MIDP 2.0 Serial API. It requires the use of a set of Hayes AT (Attention) Modem Commands to be executed from the external device to the phone in order for interface to be opened. A simple T720i test for a developer to perform for this is to start Hyperterminal with a RS232 cable connected to the phone. Once connected, open the serial connection from Hyperterminal and then use the following commands.

AT

>OK (response from T720i modem)

AT+MODE=19

(Audio beep heard as port is opened)

After this is finished the MIDlet can start opening a connection to the interface and communicating through it from with J2ME.

2. Only one application may be connected to a particular serial port at a given time. A java.io.IOException is thrown if an attempt is made to open the serial port with Connector.open() if the connection is already open.

3. Please note that the system.getProperty() parameter values have changed to access the serial ports between the different handsets.

MIDP 1.0: T720i - System.getProperty("serialport.name"), System.getProperty("serial.maxbaudrate")

MIDP 2.0: V600, C650 and onwards - System.getProperty("microedition.commports"), and use getBaudRate() method to get the baudrate.

4. The implementation of the CommConnection API does not support two serial connections in parallel.

5. Comm security: Access to serial ports is restricted to prevent unauthorized transmission or reception of data. The security model applied to the serial I/O connection is defined in the implementing profile. The security model may be applied on the invocation of the `Connector.open()` method with a valid serial port connection string. Should the application not be granted access to the serial port through the profile authorization scheme, a `java.lang.SecurityException` will be thrown from the `Connector.open()` method. The security model MAY also be applied during execution, specifically when the methods `openInputStream()`, `openDataInputStream()`, `openOutputStream()`, and openDataOutputStream() are invoked.

6. There are some serial cables that can be used. You can use serial cable RS-232 or USB. RS-232 is commonly used and some of our new released handsets like the C650 and onwards support using the serial API over the USB cable.

7. Logical port names used in the URL can be as follows for real handsets:

- COM#, where COM is for RS-232/USB ports and # is a number assigned to the port (this port name is commonly used for most of your J2ME handsets)
- IR#, where IR is for IrDA IRCOMM ports and # is a number assigned to the port (some handsets can use IrDA for connection such as A760)
- USB#, where USB is for USB ports and # is a number assigned to the port (some handsets use such port name such as E680)

So it is recommended that you use **`System.getProperty("microedition.commports")`** to discover available comm ports (MIDP2.0).

8. You can find some FAQs about serial API on motocoder site. You can search them by keywords. Here list some for your reference:

>Title: Using the Serial Port API (includes a complete sample code of using serial I/O API)

Link: http://motocoder.custhelp.com/cgi-bin/motocoder.cfg/php/enduser/std_adp.php?p_faqid=305&p_created=1074686917

>Title: Serial Port Access Motorola T720i device

Link: http://motocoder.custhelp.com/cgi-bin/motocoder.cfg/php/enduser/std_adp.php?p_faqid=79&p_created=1060598607

>Title: Descriptor Serial Port T720i

Link: http://motocoder.custhelp.com/cgi-bin/motocoder.cfg/php/enduser/std_adp.php?p_faqid=464&p_created=1096915026

Please refer to API Matrix of our handsets in the SDK documentation for details on which handsets support serial API.

Conclusion

Using serial I/O API is quite easy within J2ME, you only need to pay attention to the enclosed notes and considerations for using serial I/O, and then you can develop J2ME applications as you need.

References

1. JSR 118, CommConnection interface description. http://java.sun.com/products/midp/

2. JSR139, StreamConnection interface description. http://java.sun.com/products/cldc/

3. The Generic Connection Framework
http://developers.sun.com/techtopics/mobility/midp/articles/genericframework/